Ruby - Feature #10594

Comparable#clamp

12/12/2014 09:45 PM - findchris (Chris Johnson)

Status:	Closed
Priority:	Normal
Assignee:	
Target version:	

Description

This is basically a re-opening of the feature request of issue#4573 (https://bugs.ruby-lang.org/issues/4574), which was closed due a naming debate.

It seems the standard naming for restricting a number to a specified range is indeed 'clamp'. (1)(2)(3)

As such, can we use Yusuke Endoh's original patch with the naming adjustments? If so, I can provide accordingly.

Cheers.

(1) http://www.rubydoc.info/github/epitron/epitools/Numeric:clamp
(2) http://stackoverflow.com/questions/12020787/is-there-a-limit-clamp-function-in-ruby
(3) https://developer.gnome.org/glib/stable/glib-Standard-Macros.html#CLAMP:CAPS

Associated revisions

Revision d5a0b8e3cc48632d0cb99553a7aaf233b22a1eac - 08/11/2016 07:24 AM - nobu (Nobuyoshi Nakada)

Comparable#clamp

- compar.c (cmp_clamp): Introduce Comparable#clamp. [Feature #10594]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55863 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision d5a0b8e3 - 08/11/2016 07:24 AM - nobu (Nobuyoshi Nakada)

Comparable#clamp

- compar.c (cmp_clamp): Introduce Comparable#clamp. [Feature #10594]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55863 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 3ccd2a1bb262440ac72c8bb0d03bb7284451654f - 08/12/2016 03:55 AM - nobu (Nobuyoshi Nakada)

ChangeLog: fix name in r55863 [ci skip]

- ChangeLog: fix the original patch author's name to the name
  written in the patch, not "author name" in git.
  [Feature #10594]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55879 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 3ccd2a1b - 08/12/2016 03:55 AM - nobu (Nobuyoshi Nakada)

ChangeLog: fix name in r55863 [ci skip]

- ChangeLog: fix the original patch author's name to the name
  written in the patch, not "author name" in git.
  [Feature #10594]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55879 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 85512bdbde6f23334b4c20201b53539217518770 - 08/13/2016 02:04 PM - nobu (Nobuyoshi Nakada)

test_comparable.rb: fix clamp test

- test/ruby/test_comparable.rb (TestComparable#test_clamp): fix
  test. the result which is not clamped should be the receiver.
  [Feature #10594]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55890 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

Revision 85512bdb - 08/13/2016 02:04 PM - nobu (Nobuyoshi Nakada)

test_comparable.rb: fix clamp test

- test/ruby/test_comparable.rb (TestComparable#test_clamp): fix
  test. the result which is not clamped should be the receiver.
  [Feature #10594]

git-svn-id: svn+ssh://ci.ruby-lang.org/ruby/trunk@55890 b2dd03c8-39d4-4d8f-98ff-823fe69b080e

History

#1 - 12/12/2014 10:25 PM - findchris (Chris Johnson)

Example behavior:

```
234234234523.clamp(0..100)   # => 100
12.clamp(0..100)             # => 12
-38817112.clamp(0..100)      # => 0
```

#2 - 06/27/2015 09:55 AM - nerdinand (Ferdinand Niedermann)

Here's a pull request for this: https://github.com/ruby/ruby/pull/947

#3 - 06/29/2015 10:38 AM - kosaki (Motohiro KOSAKI)

```
+    double num_dbl = NUM2DBL(num);
+    double beg_dbl = NUM2DBL(begp);
+    double end_dbl = NUM2DBL(endp);
```

If num is Bignum, the above code could make data loss. I think.
Why can't we use "num"'s <, <= or <=> operator?

#4 - 06/29/2015 07:01 PM - nerdinand (Ferdinand Niedermann)

Motohiro KOSAKI wrote:

> ```
> +    double num_dbl = NUM2DBL(num);
> +    double beg_dbl = NUM2DBL(begp);
> +    double end_dbl = NUM2DBL(endp);
> ```
>
> If num is Bignum, the above code could make data loss. I think.
> Why can't we use "num"'s <, <= or <=> operator?

I guess you're right. I just didn't really know how to do that. How would I use the operators?

#5 - 06/30/2015 02:18 AM - 0x0dea (D.E. Akers)

- File num_clamp.c added

Ferdinand Niedermann wrote:

> How would I use the operators?

I've attached an implementation of num_clamp() which uses rb_funcall() to invoke the receiver's #< and #> methods. It also checks the value of exclp in order to properly handle ranges with exclusive ends, such that 3.clamp(1...3) == 2.

Edit: Whoops. rb_int_pred() should be called directly rather than invoked dynamically.

```
-    if (exclp) endp = rb_funcall(endp, rb_intern("pred"), 0);
+    if (exclp) endp = rb_int_pred(endp);
```

#6 - 06/30/2015 08:43 AM - Hanmac (Hans Mackowiak)

hm might it be a good idea to have such a function directly in Comparable too?

like "x".chomp("a".."e") #=> "e"

hm maybe have it a second way to call it with using "x".chomp("a", "e") too similar to Comparable#between?

#7 - 06/30/2015 10:26 AM - 0x0dea (D.E. Akers)

Hans Mackowiak wrote:

> hm might it be a good idea to have such a function directly in Comparable too?
>
> like "x".chomp("a.."e") #=> "e"
> hm maybe have it a second way to call it with using "x".chomp("a", "e") too similar to Comparable#between?

I think your suggestions make a great deal of sense. That #clamp should be defined in terms of #<=> makes Comparable its natural home. Additionally, if the two-argument form were the only way to call it, the implementation would pretty much be the same as for #between? but with greater information density:

```
 static VALUE
-cmp_between(VALUE x, VALUE min, VALUE max)
+cmp_clamp(VALUE x, VALUE min, VALUE max)
 {
-    if (RTEST(cmp_lt(x, min))) return Qfalse;
+    if (RTEST(cmp_lt(x, min))) return min;
-    if (RTEST(cmp_gt(x, max))) return Qfalse;
+    if (RTEST(cmp_gt(x, max))) return max;
-    return Qtrue;
+    return x;
 }
```

This approach does away with the potential for confusion introduced by exclusive ranges, made worse by the fact that "predecessor" is not well-defined for most Comparables.

#8 - 07/03/2015 07:15 AM - nerdinand (Ferdinand Niedermann)

That does make a lot of sense. I'll send another pull request.

#9 - 07/06/2015 03:26 PM - nerdinand (Ferdinand Niedermann)

I tried the basic implementation (adaption of #between?). What should we do about cases like these?

```
irb(main):001:0> 1.clamp(2, 1)
=> 2
irb(main):002:0> 2.clamp(2, 1)
=> 1
```

With #between? these make more sense, because nothing is in the Range of 2 to 1:

```
irb(main):003:0> 1.between?(2, 1)
=> false
irb(main):004:0> 2.between?(2, 1)
=> false
```

Should we return nil in this case for #clamp?

#10 - 07/07/2015 04:22 PM - 0x0dea (D.E. Akers)

In the case of min > max, the options seem to be these:

- return nil
- return the receiver
- raise an ArgumentError

Given that the first two are likely to cause something else to explode down the line, it seems best to just stop at the source of the error.

#11 - 07/08/2015 01:00 PM - nerdinand (Ferdinand Niedermann)

I think there is even another option: Swapping the min and max values if they are passed in the wrong order.

This is obviously a question of the principle of least surprise. If you compare #clamp to #between?, it makes sense to actually return something instead of raising, because #between? does the same.
But as you say, one could argue that it's best to catch the error as early as possible, so raising would be a good option.

The only thing I definitely would not agree with is returning the receiver, as that is a valid behaviour if the receiver is between min and max. I think it would lead to confusion.

#12 - 07/08/2015 04:08 PM - 0x0dea (D.E. Akers)

> I think there is even another option: Swapping the min and max values if they are passed in the wrong order.

I suspect there is little to no precedent for allowing "position-independent positional arguments". It's true that they could be tolerated and correctly handled in the case of #clamp, but doing so would likely hide some logic error elsewhere in the program.

#between? is a query method and thus observes the widespread convention of returning *only* either true or false. That nothing exists between 2 on the left and 1 on the right means #between? is correct not to raise given such input.

I confess that I only suggested returning the receiver to pad the list. It *would* potentially indicate to the caller that the requested operation couldn't be performed, but that's precisely what exceptions are for.

Silently swapping is antithetical to the very notion of positional arguments, returning nil is just going to make something else fail, and returning the receiver goes against the method's *raison d'être*.

That min > max should raise an ArgumentError seems the only logical conclusion.

#13 - 07/09/2015 07:33 AM - nerdinand (Ferdinand Niedermann)

> That min > max should raise an ArgumentError seems the only logical conclusion.

I agree. I'll send another pull request including that.

#14 - 07/09/2015 08:28 PM - nerdinand (Ferdinand Niedermann)

Here you go: https://github.com/ruby/ruby/pull/962

#15 - 07/09/2015 10:27 PM - nobu (Nobuyoshi Nakada)

- Tracker changed from Bug to Feature

#16 - 07/09/2015 11:12 PM - 0x0dea (D.E. Akers)

Ferdinand Niedermann wrote:

> Here you go: https://github.com/ruby/ruby/pull/962

The failure condition should be min > max rather than max < min. It's arguably rather silly to clamp to a single value, but the operation is nevertheless well-defined, and it would therefore be an error for #clamp to raise given equal min and max.

#17 - 07/10/2015 05:32 AM - nerdinand (Ferdinand Niedermann)

D.E. Akers wrote:

> Ferdinand Niedermann wrote:
>
> > Here you go: https://github.com/ruby/ruby/pull/962
>
> The failure condition should be min > max rather than max < min. It's arguably rather silly to clamp to a single value, but the operation is nevertheless well-defined, and it would therefore be an error for #clamp to raise given equal min and max.

That doesn't really change anything, but I updated the pull request. I added some more tests to prove that it doesn't raise given equal min and max.

#18 - 07/10/2015 03:39 PM - 0x0dea (D.E. Akers)

Ferdinand Niedermann wrote:

> That doesn't really change anything

You're right, of course. I'm not sure why I read it as "raise unless max > min". The error message is still slightly ill-worded, but everything else looks good. I hope #clamp makes it in. The sort[1] trick is nice, but it's a little too "clever".

#19 - 09/02/2015 06:12 PM - nerdinand (Ferdinand Niedermann)

- File comparable-clamp.diff added

- Subject changed from Numeric#clamp to Comparable#clamp

#20 - 09/14/2015 08:13 PM - kosaki (Motohiro KOSAKI)

I really dislike using Comparable for this method because clamp is NOT kind of compare. If it is Numeric's method, I'm OK.

#21 - 09/15/2015 05:24 AM - Hanmac (Hans Mackowiak)

Motohiro KOSAKI wrote:

> I really dislike using Comparable for this method because clamp is NOT kind of compare. If it is Numeric's method, I'm OK.

hm imo its better in Comparable than in Numeric because it does work for non Numeric objects too, like for String
thats why its better central in Comparable than in Numeric.

#22 - 09/15/2015 09:42 AM - Eregon (Benoit Daloze)

Hans Mackowiak wrote:

> Motohiro KOSAKI wrote:
>
> > I really dislike using Comparable for this method because clamp is NOT kind of compare. If it is Numeric's method, I'm OK.
>
> hm imo its better in Comparable than in Numeric because it does work for non Numeric objects too, like for String
> thats why its better central in Comparable than in Numeric.

Good luck explaining the precise semantics of clamp on String though.

#23 - 09/15/2015 09:48 AM - nerdinand (Ferdinand Niedermann)

Benoit Daloze wrote:

> Good luck explaining the precise semantics of clamp on String though.

IMO it's not that hard to grasp:

```
call-seq:
   obj.clamp(min, max) ->  obj

Returns min if obj <=> min is less than zero, max if obj <=> max is
greater than zero and obj otherwise.

   12.clamp(0, 100)          #=> 12
   523.clamp(0, 100)         #=> 100
   -3.123.clamp(0, 100)      #=> 0

   'd'.clamp('a', 'f')       #=> 'd'
   'z'.clamp('a', 'f')       #=> 'f'
```

#24 - 09/15/2015 10:06 AM - Eregon (Benoit Daloze)

It's just that String#<=> is not consistent with String#succ but indeed it's not so much of a problem here.

#25 - 09/26/2015 11:45 AM - nerdinand (Ferdinand Niedermann)

Where do we go from here? Is my patch acceptable? If no, what should I change? If yes, what's the process for getting it merged?

#26 - 07/21/2016 01:10 PM - nerdinand (Ferdinand Niedermann)

Anyone there? I'd love to see this in Ruby finally and I'm not the only one...

#27 - 07/22/2016 09:41 AM - duerst (Martin Dürst)

On 2016/07/21 22:10, nerdinand@nerdinand.com wrote:

> Issue #10594 has been updated by Ferdinand Niedermann.
>
> Anyone there? I'd love to see this in Ruby finally and I'm not the only one...

One thing I can suggest for those who want to see a reply to this (and other) issues is to list it on the next Developers Meeting. The last one was this week (see

https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20160719Japan
and
https://docs.google.com/document/d/1nu4pzK9nYNaKKNHQetP2xthvCVXexIsg6GUPoJR7CPQ/pub).

You should list the issues you care about under the "From non-attendees"
section.

The next meeting is scheduled for August 9th, but the wiki page for it
is not yet up.

#28 - 07/22/2016 09:47 AM - shyouhei (Shyouhei Urabe)

Martin Dürst wrote:

> The next meeting is scheduled for August 9th, but the wiki page for it
> is not yet up.

Here you are: https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20160809Japan

#29 - 07/22/2016 09:52 AM - nerdinand (Ferdinand Niedermann)

Shyouhei Urabe wrote:

> Martin Dürst wrote:
>
> > The next meeting is scheduled for August 9th, but the wiki page for it
> > is not yet up.
>
> Here you are: https://bugs.ruby-lang.org/projects/ruby/wiki/DevelopersMeeting20160809Japan

Thanks, but I don't think I have access to editing this page. Or at least I can't find the button for it...

#30 - 07/22/2016 10:55 AM - shyouhei (Shyouhei Urabe)

Ferdinand Niedermann wrote:

> Thanks, but I don't think I have access to editing this page. Or at least I can't find the button for it...

Added a link to this issue.

#31 - 08/09/2016 05:55 AM - matz (Yukihiro Matsumoto)

I accept the idea of #clamp as Comparable#clamp(min, max).
It would not accept ranges (for now).

Matz.

#32 - 08/11/2016 07:24 AM - nobu (Nobuyoshi Nakada)

- Status changed from Open to Closed

Applied in changeset r55863.

Comparable#clamp

- compar.c (cmp_clamp): Introduce Comparable#clamp. [Feature #10594]

Files

num_clamp.c	427 Bytes	06/30/2015	0x0dea (D.E. Akers)
comparable-clamp.diff	2.52 KB	09/02/2015	nerdinand (Ferdinand Niedermann)